Using Assessment Software to Create a Dialogue-Based Tutorial

O'Neill, I. (2018). Using Assessment Software to Create a Dialogue-Based Tutorial. *ACM Inroads*, *9*(1), 38-44. https://doi.org/10.1145/3130877

Published in:
ACM Inroads

Document Version:
Peer reviewed version

Using Assessment Software to Create a Dialogue-Based Tutorial

Ian O'Neill

Abstract

Automated tutorials enable students to prepare and revise topics whenever they wish. Very large and growing class sizes on key pathways mean that busy tutors and students are glad of software programs that automate some of the more routine student-tutor interactions. I present a worked example of a short automated adaptive tutorial, inspired by human-to-human dialogue and created using the kind of widely deployed assessment software that is used for authoring multiple-response (check-box) exam questions. Combining their teaching experience with some basic control logic, tutors can use to good educational effect the distinctive answer combinations that students provide.

Introduction

The aim of this article is to examine how assessment software – Questionmark[1] being the authoring tool used here – can be used to create a learning experience that resembles the traditional person-to-person university tutorial. In view of the very large class sizes in institutes of higher education, a suitable means of automating at least some of the more routine student-tutor interactions is likely to receive a warm welcome, from lecturers and students alike. A software-based tutorial that deals with run-of-the-mill issues has the potential to free tutors' and students' time for discussion of more challenging, and probably more interesting, topics. While it is currently not possible to recreate – convincingly at any rate – through software the free-flowing conversations that take place in the small tutorial group, it *is* possible with present technology to simulate some of the more predicable tutor-student interactions, especially when a well-known topic is being rehearsed.

Of course, the idea of using computers to assist students' learning is not new. Efforts in this area have a well documented history. The resulting literature is too broad and deep for detailed consideration in this article. Suffice it to say that there are traceable lines of descent from the first manifestations of mechanical multiple-choice devices in the 1920s [12], through the emergence, in the mid-20th century, of electronic digital computing and a growing recognition of its potential for Computer-Assisted Instruction (e.g. [14]), to the application of techniques from Artificial Intelligence [3], and so to the development of applications that fall under the general heading of Intelligent Tutoring System (ITS). In recent years many sophisticated ITS solutions have been developed. Some notable approaches include use of: customized study plans based on student learning profiles [8]; affect-enabled agents that motivate the student [17], or actively recognize and adapt to the student's emotional state [1]; speech recognition and dialogue modeling [13]; and cognitive models – which underpin at least one commercially released family of intelligent applications [4].

The solution proposed here, however, is quite a pragmatic one: it taps directly into the subject knowledge of the experienced tutor and his or her ability to provide appropriate responses to predictable answer combinations from the student. Given that it uses a well-established assessment package, and some straightforward control logic (even if it is applied quite creatively), tutors may be encouraged to experiment with an automated tutorial of their own. If this article stimulates such interest, it has served its purpose.

[1] Questionmark is a registered trademark of Questionmark Computing Limited, whose corporate office is located in Trumbull, Connecticut.

Key to recreating the flavor of the person-to-person tutorial is software that, like the experienced tutor, correctly interprets a combination of responses from the student to a single stimulus: 'What can you tell me about topic X?' A pattern of responses often reflects strengths and weaknesses that are shared by a number of students in a cohort, and the perceptive tutor will often recognize instinctively the aspect of the topic that is in need of attention. This is where some of the more powerful features of a package such as Questionmark can be put to good use.

In my own specialist area, software engineering, one activity that provokes predictable errors is representation of system requirements in the notation known as the *Unified Modeling Language*, or UML [2] – more about this presently. In the exercise reported here, my aim was to discover how Questionmark could, through dialogue-like exchanges, help instruct students in some fundamentals of UML, and, in particular, how the system could be made to respond appropriately to some typical combinations of errors from the student. Just as an experienced tutor coaches the student, the software must be able to provide 'feedback' that allows the student to focus on the area for improvement and to understand how to correct his/her mistakes. The examples selected for this experiment are simple ones, but illustrate techniques that can be extended to other, more nuanced tutorial topics. Figure 1 shows how the tutorial appears to the student: a UML diagram, a very open-ended question, and a selection of possible responses, some of which the student has already selected and submitted. The instance shown is an unproblematic one – the student has chosen all the correct answers and, in the text at the bottom of the screen, is duly congratulated!

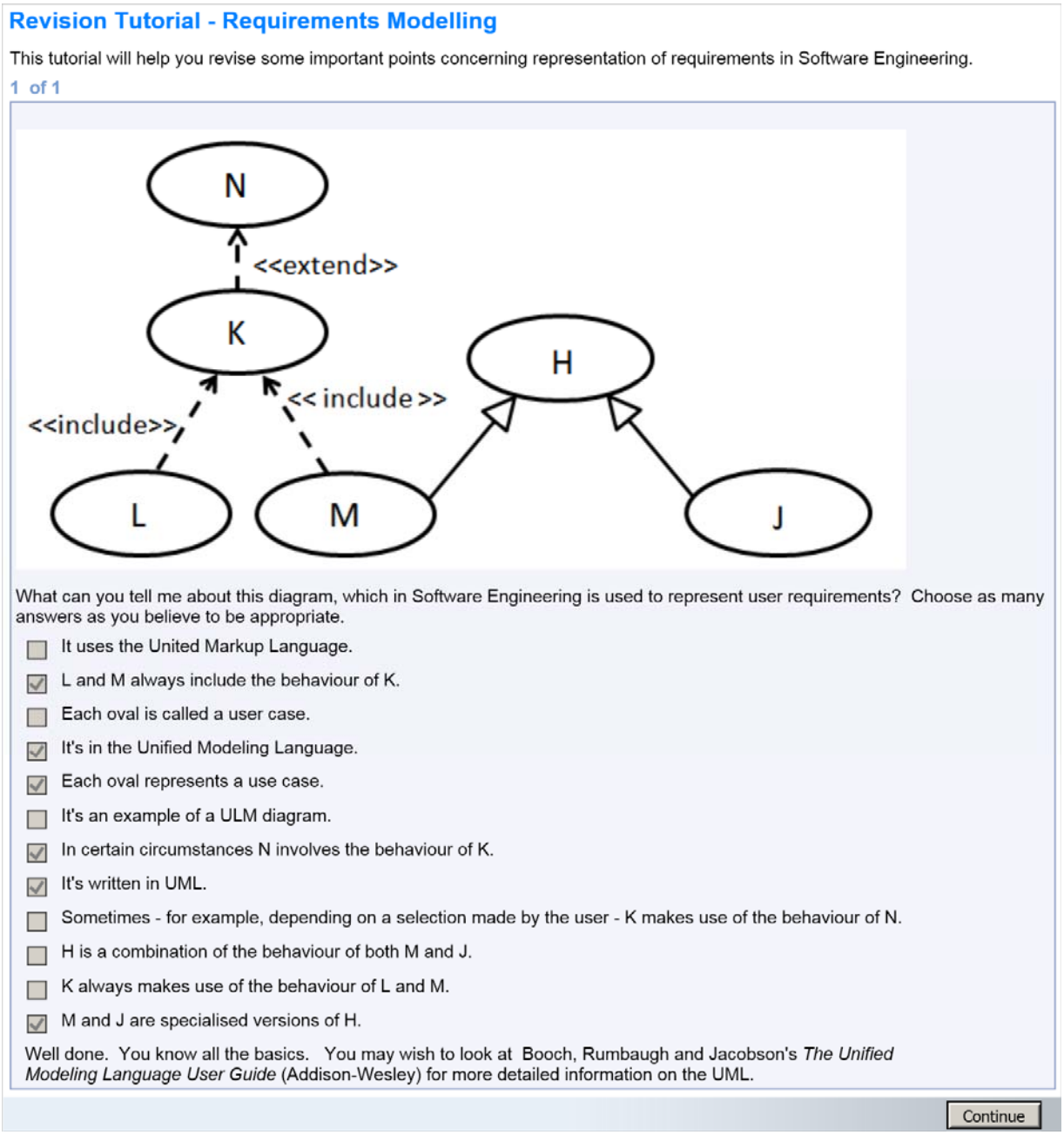

Figure 1. How the tutorial looks: A correct set of answers, and congratulatory remarks.

What is Questionmark?

Questionmark is a widely-used software package for authoring, conducting and marking on-line assessments. Instructors often use Questionmark to create simple tests made up of 'multiple choice' questions (in which radio buttons allow one answer only), or else 'multiple response' questions (where check-boxes accommodate several possible answers). The system allows the assessor to assign marks to each possible choice or response, so that, when the test is taken, the system returns the score of each student in the cohort, along with – if it is required – a full

report (for tutor and/or student) on how the student answered. Typically students complete such tests by answering the questions in the sequence in which they appear, making sure to complete the test within the time limit, and taking the test only once if they achieve a good enough score first time round. However, Questionmark is capable of more than this. [6] enumerates its many question types, while, in a survey of twenty computer-assisted assessment (CAA) applications [5], Questionmark's large feature set compares favorably with that of rival products. It has been used over the years as a means of studying the effectiveness of CAA [7], and more specifically the effect of *feedback* in computer-based assessment (CBA) [16]. Sometimes it is even used to assess the performance of students who have used adaptive tutorials that have been implemented in other software [15]. However, for the novel, dialogue-like behaviour described here, it is particularly helpful that Questionmark allows assessment authors to give questions a variety of *named outcomes* (which correspond to different answers or answer combinations that the student might select and that the author regards as significant) and that these outcomes may be used to influence the path the student takes through the test or exam – or in this case, the tutorial.

Using the assessment system to support text-based human-computer dialogue

How might one adapt the sequence of questions – and the feedback associated with them – to outcomes resulting from different (combinations of) responses/choices from the student? In the terminology used in dialogue research [10,11], which has influenced my approach to technologies for learning and teaching, this amounts to a simple 'frame-based approach' [9]: key values input/uttered by the user are matched against a 'frame' of features that the dialogue designer expects to occur in different combinations according to circumstance. In Figure 2 we see examples of such dialogue frames, in the context of a system-student interaction: each frame consists of a selection of answer options, some of which are correct and some incorrect (these options can be regarded as 'features'). The combination of features and values that actually occur are used as conditions that determine the system's next 'move', or step, in the dialogue. In Figure 2 the values are shown as 1s and 0s, representing respectively the answers that the student selected and the answers that he or she did not select. For a limited number of *named outcomes* (which correspond to the most significant answer combinations), and a small number of possible steps in the student's path through the questions, the required authoring technique is fairly straightforward. However, as the required system behavior comes to resemble that of a human-to-human dialogue, it is important to consider a more varied range of outcomes, any of which might be used to trigger a jump to an appropriate block of follow-on questions. More nuanced tutorial exchanges will, therefore, require more subtle design of questions, outcomes and conditional jumps. In **A dialogue management strategy** below**,** we consider in greater detail some application-specific rules that account for the dialogue steps in Figure 2.

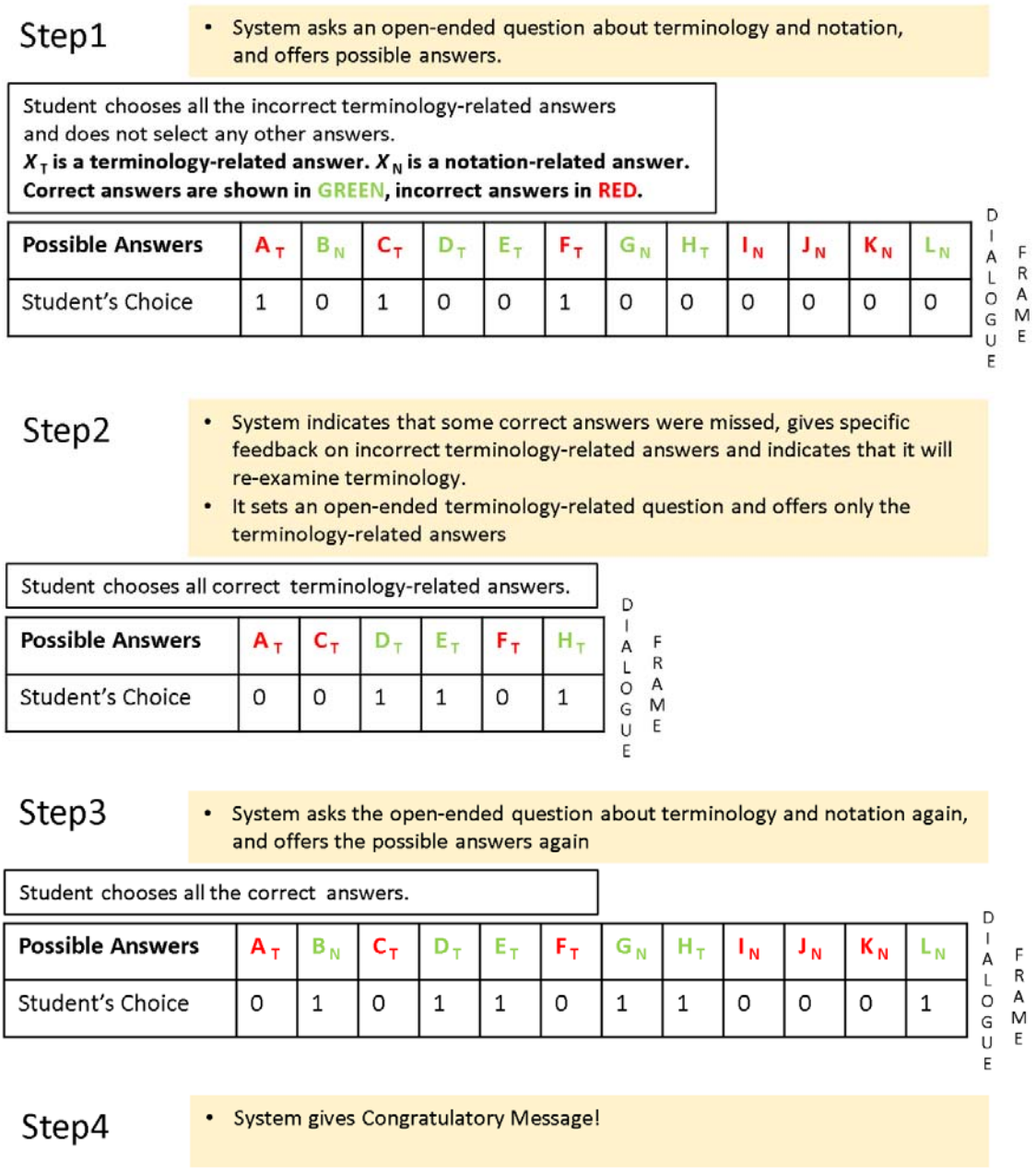

Figure 2. A frame-based dialogue fragment, managed according to the rules outlined in the section **A dialogue management strategy**. The 1s and 0s represent respectively the answers that the student selected and the answers that he or she did not select.

In defining a *named outcome*, the Questionmark author gives a name to, and uses a simplified form of Boolean logic to describe, the most significant combinations of 'answers given' and 'answers omitted' by the student. In logical terms, the outcome is described as an ANDing or ORing of answers given by the student, or a negation of possible answers that the student did not give. The author can then use the occurrence of certain *named outcomes* to make

the system's dialogue with the student 'jump' to a new block of follow-on questions. (We shall look more closely at a specific example of such a *named outcome* – one called *Terminology_All_Wrong* – in the following section.) Alternatively, accumulated scores can be used as criteria for jumping to a new set of questions or for ending the tutorial.

Outcomes may be associated with feedback – perhaps a few words of acknowledgement, advice or encouragement from the instructor – or, they may result in a score being awarded. A single question (or *stimulus*) may have more than one outcome. This opens the possibility of a single question with several pieces of feedback. On the other hand, some outcomes may be declared as stopping points: once the condition for the outcome is satisfied, no further outcome is tested. Furthermore, outcomes may be quite subtle, allowing the author of the automated tutorial to distinguish between a student's ability in different areas of knowledge within a broader topic.

A dialogue management strategy

Take a problem that requires students to give answers in *two* related areas of knowledge – for example, interpreting a symbolic *notation*, and using the correct *terminology* to describe the notation (we shall put this in a subject-specific context in a moment). In these circumstances, depending on the tutorial style that the author of the system wishes to adopt, possible conditions arising in the student's interaction with the system, and the ensuing dialogue steps, might include the following points. Though informally expressed here, these points can be regarded as a set of dialogue management *rules* that collectively impose a rule-based dialogue management *strategy*.

Dialogue Management Rules

1. For a student who is able to solve the problem correctly – choosing all the correct answers and none of the incorrect ones – the system provides appropriate words of congratulations and the tutorial concludes.
2. For a student whose knowledge is sporadic (a low score across the main knowledge areas – in this instance, *notation* and *terminology*), the system provides feedback and presents the whole exercise to the student again.
3. For a student who achieves a reasonable score overall, the system provides feedback and a 'could do better' message, and the tutorial concludes.
4. For students who fail the test completely, by providing no correct answers and all the wrong ones, feedback is provided, and the test is split into its component areas, each area being tested separately (in the manner described next).
5. For a student who completely fails in an area (completely misinterpreting the *symbols*; or using all the incorrect and none of the correct *terminology*), the system gives feedback on the student's previous responses and poses a more focused question that deals with the failed component. Each of these focused, area-specific follow-up questions has a correspondingly restricted set of possible answers. The failed component is repeated, with feedback each time, until the student achieves a satisfactory performance, and then the second failed area (if there is one) is similarly addressed, before the whole tutorial is repeated – giving the student another opportunity for a 'clean run through'.
6. If a student makes no attempt to answer a question posed by the system (a blank response), the system repeats the question.

Figure 2, introduced previously, represents a small segment of a dialogue that is managed according to these rules. In Figure 2, Steps 1 – 3, the student's incorrect terminology-related answers, the system's more restricted follow-up question, and the system's return to the original question correspond to Rule 5 of the sample dialogue management strategy above.

Implementing this behaviour requires use of *named outcomes* and Boolean conditions that describes those outcomes. Questionmark provides a GUI-based 'outcome editor' and with it a mark-up language, QML, that supports a basic set of logical operators. Let us focus on one of these *named outcomes* – we shall give it the name *Terminology_All_Wrong* – an outcome associated with Step 1 of Figure 2. The assessment author uses the mark-up language to define *Terminology_All_Wrong* as a specific combination of 'responses selected' and 'responses not selected' by the student: in this case, the definition – if we use the same letters as are shown in Figure 2 – corresponds to the Boolean formula

$$A_T \wedge C_T \wedge F_T \wedge \neg D_T \wedge \neg E_T \wedge \neg H_T$$

i.e. the student has selected all the incorrect terminology-related answers and none of the correct terminology-related answers. The author specifies the feedback that the system should display when the *named outcome* or condition occurs. Then the author structures the overall flow of the tutorial in such a way that, if this *named outcome* is true, the system will 'jump' to the appropriate follow-up question (in this case the terminology-focused question). In the section **How does the solution run?** we see the effect of this and other *named outcomes* on the system feedback and the dialogue flow.

Steps 3 and 4 of Figure 2, where the student selects all the correct answers, and the system generates a congratulatory message, corresponds to Rule 1 of our Dialogue Management Strategy.

A sample problem

For a trial run of the approach, I chose a specific example: the interpretation of UML use case diagrams, an instance of which we have already seen in Figure 1. *The Unified Modelling Language* (UML) *User Guide* 2nd Edition [2] was used as a reference. 'Use cases' represent sets of sequences of actions (typically user-computer interactions), that provide some outcome of value to a user of a computer system. They help give structure to the very early stages of the software development process, and they indicate observable behaviour that should eventually be tested from a user perspective. The *use case diagram* models the requirements of a software system as inter-connected use cases. Having used and taught a use case approach to systems analysis for many years, I am usually on the lookout for a number of very basic misunderstandings. These concern:-

- the definition of a use case ("a description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor" [2:228]), and UML-related vocabulary more generally (matters of *terminology*);
- confusion of <<include>> and <<extend>> arrows (here I treat this as a question of *notation*) – <<include>> arrows being used to show behaviour that is always included in the flow (the sequence of user-computer interactions) of another use case, and <<extend>> arrows for behaviour that is invoked only conditionally or exceptionally;
- getting the sense of the unidirectional <<include>> arrows, <<extend>> arrows and open-headed (white-headed) generalization arrows 'the wrong way round' – again a question of *notation*. The <<include>> and <<extend>> arrows are read in the direction in which they point (e.g. Use Case K 'extends' [under certain circumstances] UseCase N'; Use Case L always 'includes' Use Case K). A generalized use case appears at the head of an open-headed arrow, its more specialized variant at the tail.

The tutorial example that I created makes few prior assumptions about the student's knowledge of UML. It simply presents the student with a moderately challenging example of the UML use case notation in action, and asks the student to select the correct statements from the options available. Students can choose as many or as few answers as they think appropriate. As just described, the possible answers test two main areas of knowledge, *notation* and

terminology, the same areas that we used in our earlier, more general example of dialogue flow – though students are not explicitly told the areas being tested as they start the automated tutorial.

How does the solution run?

The screen shots illustrate some of the dialogue-like interaction between the student and Questionmark in the role of tutor. The opening screen (similar to that shown in Figure 1, but with the answer boxes unchecked, and with a Submit button for the student to click) provides the stimulus that starts the 'conversation' between student and system. The use case diagram in the opening screen, even though it is a simple one, includes an example of each of the possible relationships between the use cases (represented by ellipses in the diagram). Included in the mix of possible responses are correct answers that allow the student to give a full account of the diagram – describing correctly the nature of the relationships represented by the symbolic *notation*, and correctly using use-case-related UML *terminology*. A student who provides a perfect set of responses (as is the case in Figure 1) is congratulated appropriately and directed towards some relevant further reading: clearly this tutorial is well within the student's capabilities; they can afford to expand their knowledge!

By contrast an inaccurate and incomplete set of responses to the opening stimulus leads to the system comments shown in Figure 3. Some of the comments (those marked 1 and 2) are feedback resulting from *named outcomes* with their Boolean conditions: i.e. from the answers available, the student overlooked at least one correct use of *terminology*, or missed at least one correct interpretation of the direction of an arrow or its label (a matter of *notation*). Comments marked 3, 4 and 5 in Figure 3 represent feedback resulting directly from the individual responses selected by the student.

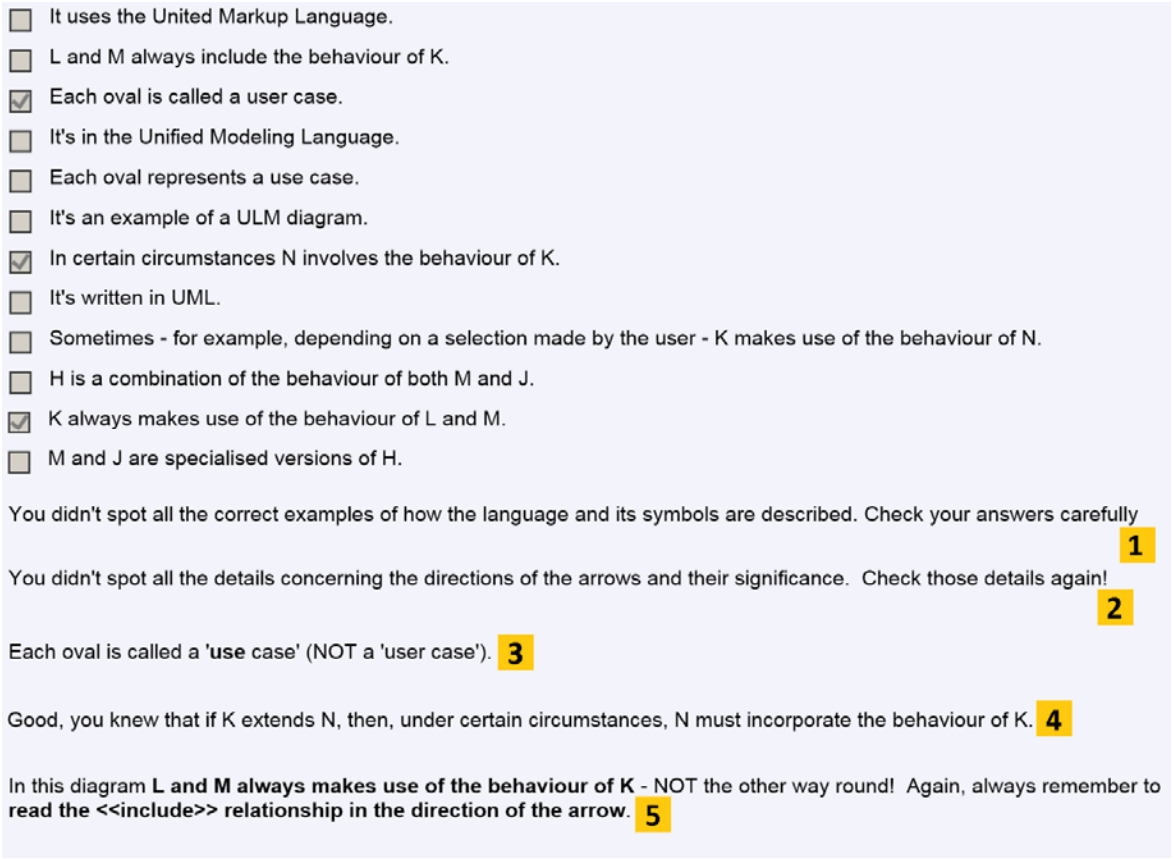

Figure 3. An inaccurate and incomplete set of responses from the student, and the tutorial system's comments. (Annotated lines are referenced in the main text.)

Overall the student fared poorly in the responses recorded in Figure 3 – and, in this implementation, the system has a measure of this, because even though it does not show the student the scores that result from the responses, it does keep a tally of scores: 1 for a correct response; -1 for an incorrect response. The poor overall performance has a further consequence. With some words of explanation – "You could do much better on these answers – so let's try again..." (these words come in the form of a message box (not shown here) that requires only an acknowledgement (a 'submit') from the student) – the system now brings the student back to the opening question, with its full set of possible responses, so that the student may choose from these again.

As has been indicated earlier, students who choose all the incorrect responses in either or both of the two (implicit) subject areas provoke a more interesting system reaction. Figure 4 shows the system's feedback when the student chooses all the incorrect terminology-related answers, and does not select any other answers (this is the student input that is shown in more abstract form in Step 1 of Figure 2, and that satisfies the condition of the *named outcome Terminology_All_Wrong*). The last three lines in Figure 4 are triggered by the student's individual 'check-box' responses: here we are reminded of the responses that should have been chosen (and the responses that the student actually chose). The two comments 'You didn't spot all the correct examples […]', and 'You didn't spot all the details […]' are feedback from *named outcomes* and we have seen this feedback already in Figure 3. The short block of three lines beginning 'The terminology caused problems here' is feedback triggered by *Terminology_All_Wrong*, the *named outcome* we looked at in some detail in the previous section: not only does the feedback for this outcome include a summary of the terminology that the student should have recognized

("Remember: the diagram is about 'use cases' and it's written in the Unified Modeling Language, or UML"), but it also lets the student know that system will be returning to this area specifically ("We'll try the naming questions again in a moment"). And so, once the student acknowledges the system's comments by clicking 'Continue', the system generates a single stimulus that deals only with terminology. The new, more focused question is shown in Figure 5.

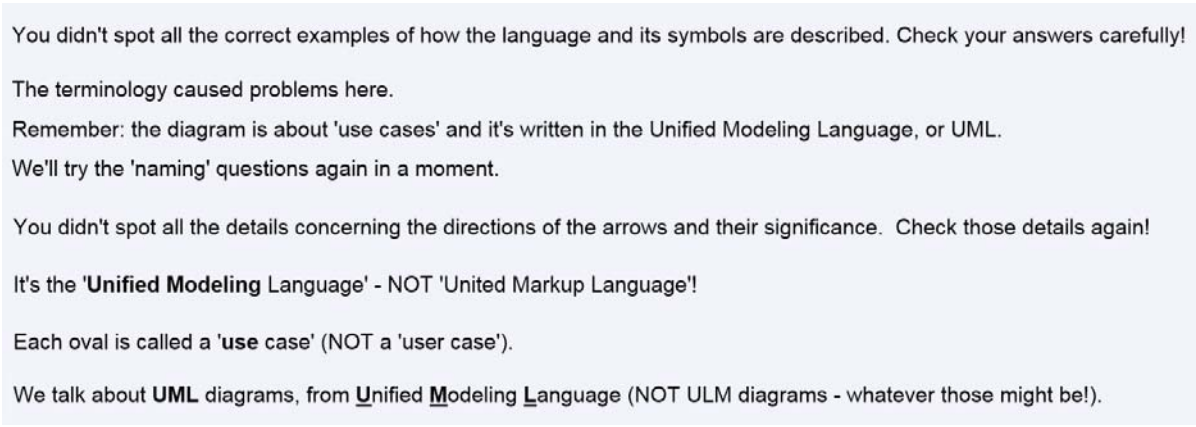

Figure 4. The student selected all the incorrect terminology-related answers and did not choose any other answers: now the system responds (cf. Figure 2, Steps 1 & 2).

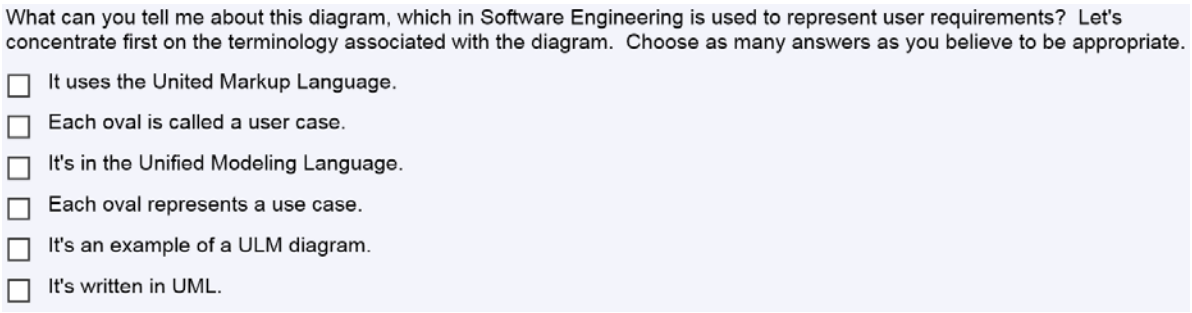

Figure 5. The system's follow-up to the problems noted in Figure 4 (cf. Figure 2, Step 2).

How do the students react?

The initial reaction to the system (and at this stage the author can claim no more than an initial reaction) has been very positive – to judge at least from the feedback given by those students who took a few moments to complete a short user survey that was also implemented in Questionmark. The great majority of students thought the tutorial was helpful, and expressed the wish that such tutorials should be longer and on a wider selection of topics. The most common dislike was that the tutorial was too short, while a small number expressed frustration that the 'full solution' wasn't simply provided after a number of imperfect attempts. There were also a few references to awkward wording in the system's feedback (for the teacher there is sometimes a compromise between natural phrasing and unambiguous technical accuracy), and awkward navigation though Questionmark.

While the 'dislikes' were quite restrained in tone, the 'likes' expressed obvious enthusiasm, giving a strong impression that the students were generally well disposed towards the approach.

"It tested a vast range of knowledge within UML. It would be really beneficial to have a quiz [tutorial] on each topic."

Or, even more succinctly:

"Brilliant idea – wish there was one for each topic."

However, comments that took up the theme of feedback were perhaps the most telling, given the ambition of this exercise to create a system that would engage in an adaptive dialogue with the user. Here are a few especially pertinent remarks...

"When you give an incorrect answer you actually got feedback and the reason why your answer is wrong, helping my understanding."

"It allowed me to get in depth feedback on the question, so I knew where I had gone wrong."

"The variety of answers made it more challenging than a typical multiple-choice question which helps during revision to gain a greater understanding than the exam would delve into."

...And the following in particular:

"It was specialized in its feedback and provided the option to repeat questions to help solidify understanding (in the context of Questionmark style questions)"

This is perhaps the system's greatest advantage: it gives feedback appropriate to the student's responses, focuses on the area where reinforcement is needed, and manages to do that in the familiar context of the automated assessment tool Questionmark.

The students' reaction to the experiment seems to confirm that they perceived a benefit. Indeed, students who had responded to the Feedback Questionnaire (presumably – if they were undertaking the exercise in good faith – after having tried the tutorial), also performed noticeably better in the relevant exam questions than non-respondents. Of course, this 'better performance' cannot be directly ascribed to the on-line tutorial: such a claim would require a more closely controlled experiment than the initial field trial reported here. But the signs are promising.

Taking stock and looking ahead

My initial experience of creating and deploying an automated, dialogue-based tutorial in a 'live' teaching-and-learning environment has been very positive, but the process is not without its challenges.

The authoring tool used, Questionmark, is an effective and well-proven means of reaching and managing large cohorts of students for on-line assessment – and these considerations are of overwhelming importance to the many large institutions that rely on the software. However, since is not a specialized tool for building human-computer natural language dialogues, using it to achieve the kind of outcomes described here requires some perseverance.

For instance, a more specialized dialogue tool (and generally such tools are for spoken dialogue, which has extra complications of its own) might be expected to offer the convenience of drag-and-drop dialogue states and mouse-drawn transitions between states [9:164 ff.] – such aids to dialogue design are not available in the software used here.

In more general terms, as is always the case with dialogue development, special care must be taken with the wording of questions, acceptable answers and feedback. It is particularly important that each piece of feedback conveys the

correct message, with the right tone, whenever it occurs, either as a phrase in isolation or in combination with other phrases. Much testing, restructuring of dialogue flow and rewording is to be expected, for a dialogue system has to convey important information correctly and clearly: in this case, information that a student may be relying on as part of their exam preparation.

Despite the challenges, we hope to develop more tutorials of the kind described here, in my own discipline (software engineering) and in other disciplines. The students' wish for more and longer automated tutorials is an incentive; so too is the interest expressed by colleagues – teachers in my own and other departments. However, as well as meeting immediate development requests, we now want to study, in a more controlled manner, the pedagogical effects of the on-line, dialogue-based tutorial, specifically the usefulness of its adaptive elements, which require the greatest time and effort to implement well. We wish to examine different approaches to implementing such systems, in terms both of methodology and technology: how to create appropriate computer-based tutorials most efficiently; and how to use them to greatest benefit in teaching programs. As new, enhanced and more versatile means of implementing such systems become available, I and my colleagues will be eager to assess the opportunities they offer. And, as software engineers, we shall also be watching out for opportunities to create some of our own, novel in-house solutions.

References

1. Banda, N. and Robinson, P. Multimodal Affect Recognition in Intelligent Tutoring Systems. in *Affective Computing and Intelligent Interaction 2011, Lecture Notes in Computer Science*, 6975, edited by S. D'Mello, A. Graesser, B. Schuller, and J.-C. Martin (Berlin, Heidelberg: Springer, 2011), 200-207.
2. Booch, G., Rumbaugh, J. and Jacobson, I. *The Unified Modeling Language User Guide.* 2nd ed. (New Jersey: Addison Wesley, 2005).
3. Carbonell, J.R. AI in CAI: an artificial intelligence approach to computer-assisted instruction. *IEEE Transactions on Man-Machine Systems*, II (1970), 190–202.
4. Carnegie Learning; http://www.carnegielearning.com. Accessed 2017 April 4.
5. Fuentes, J.M., García, A.I., Ramírez-Gómez, Á. and Ayuga, F. Computer-Based Tools for the Assessment of Learning Processes in Higher Education: A Comparative Analysis. in *8th International Technology, Education and Development Conference.* (Valencia, Spain, 2014), 976–984.
6. Ganci, J. Tool Overview: Questionmark Perception. *Learning Solutions Magazine,* February 22, 2010; https://www.learningsolutionsmag.com/articles/420/tooloverview-questionmark-perception. Accessed 2017 August 17.
7. Harrison, M., Green, D., Pidcock, D. and Palipana, A. HELM Educational Transfer. *MSOR Connections* 7, 1 (2007), 20–22; http://icse.xyz/mathstore/headocs/Harrisonetal.pdf. Accessed 2017 August 17.
8. Keleş, Aytürk, Ocak, R., Keleş, Ali and Gülcü, A. ZOSMAT: Web-based intelligent tutoring system for teaching-learning process. *Expert Systems with Applications*, 36 (2009), 1229–1239.
9. McTear, M.F. *Spoken Dialogue Technology: Toward the Conversational User Interface.* (London: Springer-Verlag, 2004).
10. O'Neill, I., Hanna, P., Liu, X., Greer, D. and McTear, M.F. Implementing advanced spoken dialogue management in Java. *Science of Computer Programming*, 54, 1 (2005), 99–124.
11. O'Neill, I., Yue, A., Liu, W. and Hanna, P. Adaptive dialogue strategy selection through imprecise probabilistic query answering. *Lecture Notes in Artificial Intelligence*, 6717 (2011), 675–687.
12. Pressey, S.L. A simple apparatus which gives tests and scores – and teaches. *School and Society*, 23 (586) (1926), 373–376.
13. Rus, V., D'Mello, S., Hu, X. and Graesser, A.C. Recent Advances in Conversational Intelligent Tutoring Systems. *AI Magazine*, 34, 3 (Association for the Advancement of Artificial Intelligence, 2016), 42–45.
14. Suppes, P., Jerman, M. and Brian, D. *Computer-assisted Instruction: The 1965–66 Stanford Arithmetic Program*. (New York: Academic Press, 1968).
15. Van Es, S.L., Kumar, R.K., Pryor, W.M., Salisbury, E.L. and Velan, G.M. Cytopathology whole slide images and adaptive tutorials for postgraduate pathology trainees: a randomized crossover trial. *Human Pathology*, 46 (2015), 1297–1305.

16. van der Kleij, F.M., Eggen, T.J.H.M., Timmers, C.F. and Veldkamp, B.P. Effects of feedback in a computer-based assessment for learning. *Computers & Education*, 58 (2012), 263–272.
17. Woolf, B.P., Arroyo, I., Muldner, K., Burleson, W., Cooper, D.G., Dolan, R. and Christopherson, R.M. The Effect of Motivational Learning Companions on Low Achieving Students and Students with Disabilities. in *Intelligent Tutoring Systems 2010, Lecture Notes in Computer Science*, 6094, edited by V. Aleven, J. Kay and J. Mostow (Berlin, Heidelberg: Springer, 2010), 327–337.

Ian O'Neill

School of Electronics, Electrical Engineering and Computer Science
Queen's University
Belfast
BT7 1NN
Northern Ireland

i.oneill@qub.ac.uk